REMARKS

Claims 1-23 are all the claims presently pending in the application. Claims 1 and 23 have been amended to more particularly define the invention.

It is noted that the claim amendments are made only for more particularly pointing out the invention, and not for distinguishing the invention over the prior art, narrowing the claims or for any statutory requirements of patentability. Further, Applicant specifically states that no amendment to any claim herein should be construed as a disclaimer of any interest in or right to an equivalent of any element or feature of the amended claim.

Claims 21-22 stand rejected under 35 U.S.C. §112, first paragraph as allegedly not enabled by the specification.

Applicant notes that claim 22 is not subject to any prior art rejection and would, therefore, be presumably allowable except for the alleged informalities.

Claims 1-13 and 17-21 stand rejected under 35 U.S.C. §102(b) as allegedly being anticipated by Mellor-Crummey ("Compile-time Support for Efficient Data Race Detection in Shared-Memory Parallel Programs"). Claims 14-16 and 23 stand rejected under 35 U.S.C. §103(a) as allegedly being unpatentable over Mellor-Crummey in view of Flanagan et al. (U. S. Patent No. 6,343,371).

These rejections are respectfully traversed in the following discussion.

I. THE CLAIMED INVENTION

The claimed invention (e.g., as recited in claim 1) is directed to a method for statically detecting a datarace condition in a multithreaded application. The method includes inputting a set of input information, processing the set of input information by comparing threads that may execute statements in a statement pair, and outputting a statement conflict set that identifies the statement pairs whose execution instances definitely or potentially cause dataraces, without executing the multithreaded application.

Conventional methods (e.g., static datarace detection methods) identify many dataraces that may never be exhibited in any program execution (e.g,."false positives"). In addition,

programmer annotations are required to improve the effectiveness of such conventional tools (Application at page 6, lines 1-10).

The claimed invention, on the other hand, processes the set of input information by comparing threads that may execute statements in a statement pair (Application at Figure 3; page 8, line 18-page 9, line 9; page 14, lines 9-12). This feature allows the claimed method (e.g., and system) to more accurately detect a datarace condition (e.g., a condition where a datarace definitely will occur or may possibly occur) than in conventional methods (Application at page 6, lines 12-15).

II. THE 35 USC 112, FIRST PARAGRAPH REJECTION

The Examiner alleges that claims 21-22 are not enabled by the specification. Applicant submits, however, that these claims are clearly enabled by the specification.

First, Applicant would remind the Examiner that to support her allegation that these claims are not enabled by the specification, the Examiner must show that one of ordinary skill in the art could not read the specification and know how to make and use the claimed invention without undue experimentation. Here, the Examiner has clearly failed to meet this burden.

With respect to claim 21, the Examiner asserts that the specification does not describe the concept of "explicitly tagging a statement with a set of threads". However, Applicant would point out that **this phrase is a non-mathematical explanation of the following notations referred to on pages 19 and 21**:

MustThreadObj(p(S.i)) and MayThreadObj(p(S.i)),

where the former means the set of threads, tagged for statement p, that ***must*** execute p, and the latter means the set of threads, tagged for p, that ***may*** execute p. Thus, the Examiner's assertions are clearly incorrect.

Further, with respect to claim 22, the Examiner asserts that the term "lock" is not mentioned in the specification. However, Applicant would point out that the term "lock" may refer to synchronization objects which are clearly described in the specification, for example, at page 10, line 13. **Applicant respectfully submits that practitioners of multithreaded**

application developers understand well that the two terms are synonymous to each other. Computing the synchronization objects of a statement is described in detail on page 15, lines 10 to 12 of the specification. Further, "Comparing sets of locks" is clearly described in the specification, for example, on page 19, line 24-page 20, line 8, which states:

> *"The synchronization objects of two statements allow the determination of whether two statements allow the determination of whether two statements (or two executions of a single statement) executed by two different threads are synchronized with respect to each other via the synchronization objects (e.g., of synchronized methods and synchronized blocks)."*

Thus, contrary to the Examiner's allegations, claims 21 and 22 are fully enabled by the specification.

In view of the foregoing, the Examiner is respectfully requested to withdraw this rejection.

III. THE ALLEGED PRIOR ART REFERENCES

A. Mellor-Crummey

The Examiner alleges that Mellor-Crummey teaches the claimed invention of claims 1-13 and 17-21. Applicant submits, however, that there are elements of the claimed invention which are neither taught nor suggested by Mellor-Crummey.

Mellor-Crummey discloses a compile-time support system for datarace detection in shared-memory programs. Specifically, Mellor-Crummey discloses ERASER, a datarace instrumentation tool that uses program analysis to prune the number of references to be monitored (Mellor Crummey at Abstract).

However, Mellor-Crummey does not teach or suggest "*processing the set of input information by comparing threads that may execute statements in a statement pair*", as recited, for example, in claim 1 and similarly recited in claims 17, 20 and 23.

As noted above, unlike conventional methods (e.g., static datarace detection methods)

which identify many dataraces that may never be exhibited in any program execution (e.g,."false positives"), the claimed invention processes the set of input information by comparing threads that may execute statements in a statement pair (Application at Figure 3; page 8, line 18-page 9, line 9; page 14, lines 9-12). This feature allows the claimed method (e.g., and system) to more accurately detect a datarace condition (e.g., a condition where a datarace definitely will occur or may possibly occur) than in conventional methods (Application at page 6, lines 12-15).

Clearly, this feature is not taught or suggested by Mellor-Crummey. Indeed, Mellor-Crummey is completely unrelated to the claimed invention.

First, on page 2 of the Office Action the Examiner alleges that Mellor-Crummey discloses this feature of the claimed invention. However, Applicant would point out to the Examiner that the concept of a thread described in Mellor-Crummey is different from that of a Java-like object oriented program, which may be included, for example, in the claimed invention. This difference is clear from footnote 1 on page 132 in Mellor-Crummey which recites, "[w]e use the term thread to denote the basic unit of concurrency (e.g., an interation (sic) of a parallel loop body.)"

This footnote is important for Mellor-Crummey's paper because the paper deals with scientific programming in Fortran, which, unlike Java-like object-oriented languages, **has no notion of threads as defined in the claimed invention**. Mellor-Crummey simply borrows the term threads "to denote a basic unit of concurrency" such as iterations of a parallel loop body, each of which can be assigned to different computing units (e.g., CPUs) by a parallel compiler such as ParaScope. The programmer, however, has no means of expressing the relationship between two CPUs to which two different iterations should be assigned, **because there is no concept of threads that can be explicitly manipulated by the programmer in such programming languages**. In light of this, the phrase "compare threads that may execute statements in a statement pair" found in Mellor-Crummey simply means "comparing two iterations of the same parallel loop iteration."

On the other hand, in a Java-like language (e.g., which is a target of the claimed invention) a "thread" may be construed as a language construct that the programmer can

explicitly create as an object, pass to subroutines, and execute on each statement. Therefore, a statement of a program written in such a language has an explicit set of threads that execute the statement, and the program can make an explicit reference in the program to such threads executing a given statement, **which is not possible for programs supported by Mellor-Crummey**. Statically computing the set of threads that can execute a statement is referred to as "tagging a statement with a set of threads" which may be computed, as described in the application, by identifying the thread-roots from which the statement can be reached and then computing the alias sets of the "this" parameter of each of the thread-roots.

In Mellor-Crummey, however, **only iterations of the same parallel loop can execute concurrently**. Furthermore, the execution ordering of two concurrent bodies (i.e., "threads" in Mellor-Crummey's notation) are always regarded as unordered and can cause a datarace if they access the same variable (unless there are additional operations such as semaphores and locks that order the execution of part of the loop iterations, which is not addressed Mellor-Crummey). Because of this, the major mechanism Mellor-Crummey uses for his static datarace detection is data-dependence testing among iterations of the same parallel loop body, which is a well-understood prior art.

However, in Java-like object-oriented languages, any two statements that are reachable from different thread-root nodes, or that are reachable from the same thread-root node with more than one thread objects passed to its "this" parameter, can be executed by two different threads. However, the execution order of two statements by different threads can still be ordered by synchronizations (e.g., locks), which can disallow dataraces between the two statements even if they access the same variable. Thus, **unlike Mellor-Crummy which uses a dependence test applied only to iterations of the same parallel loop body, the claimed invention includes datarace detection which is applied to any two statements that may execute in parallel independently of whether they belong to the same parallel loop**.

Turning again to claim 1, the Examiner alleges that the processing "by comparing threads that may execute statements in a statement pair" of the claimed invention is disclosed in the first paragraph, section 2 on page 130 in Mellor-Crummey. **The Examiner is completely incorrect.**

Indeed, Applicant would again point out that ParaScope's notion of thread is **different from that of a Java-like language** that explicitly defines threads as programming constructs. That is, unlike a "thread" the claimed invention, a "thread" in ParaScope denotes the basic unit of concurrency (e.g., an iteration of a parallel loop body), which cannot be explicitly manipulated by the programmer.

The Examiner also alleges that Mellor-Crummey discloses "outputting a statement conflict set" as in the claimed invention at the second paragraph under section 3.2 on page 132. However, Applicant would point out that Mellor-Crummey's data dependence analysis merely deals with the variables (e.g., memory locations) accessed by **different iterations of the same parallel loop body**. A test in the claimed invention, on the other hand, may also deal with threads (e.g., as explicit programming objects) executing statements, synchronization objects used in executing statements, and statements not belonging to the same parallel loop body. Thus, the Mellor-Crummey method is completely different than the claimed invention.

With respect to claims 2-4, the Examiner alleges that Mellor-Crummey discloses these limitations on pages 130-132. However, Applicant would point out that these passages merely describe comparing the references to variables of two statements. A test in the claimed invention, on the other hand, **may also compare the explicit thread sets and the synchronization-object sets**.

With respect to claim 5, the Examiner alleges that page 131 of Mellor-Crummey discloses a set of input information which includes a multithreaded context graph. However, the Examiner must understand that Mellor-Crummy's meta-information is merely a conventional interprocedural program representation **without the concept of thread-root nodes**.

With respect to claim 6, the Examiner alleges that page 131 of Mellor-Crummey discloses an interprocedural call graph having each of a plurality of synchronized blocks as a separate node. However, Applicant would again point out that **parallel Fortran applications supported by ParaScope do not have the notion of synchronized blocks in a Java-like languages**. A synchronization block in Java, on the other hand, is a single-entry single-exit block that identifies a set of statements that are "guarded" by the same synchronization object.

ParaScope does not have this programming construct, and its program representation does not have the same concept of the claimed invention which can represent synchronized blocks as separate nodes.

With respect to claim 9, the Examiner alleges that page 132 of Mellor-Crummey discloses performing escape analysis as in the claimed invention. However, Mellor-Crummy simply assumes that any local variables passed to called procedures may be accessed in parallel, which cannot be called an escape analysis.

With respect to claim 10, the Examiner alleges that pages 132-133 of Mellor-Crummey disclose computing a node conflict set as in the claimed invention. However, again, an important difference between the claimed invention and Mellor-Crummey, is what is considered in determining conflicts. **Mellor-Crummy only considers references, while the claimed invention may also consider synchronization objects and thread objects**.

With respect to claim 11, the Examiner alleges that page 131 of Mellor-Crummey discloses initializing a synchronization object set as in the claimed invention. However, **Mellor-Crummey's program representation does not have the notion of synchronization objects nor multithreaded contexts**. Instead, Mellor-Crummey uses conventional "abstract syntax trees associated with semantic information". (e.g., see Mellor-Crummey at page 131, 1st paragraph under Section 3).

With respect to claim 13, the Examiner alleges that pages 132-133 of Mellor-Crummey disclose identifying reachable conflicting node pairs as in the claimed invention. However, Applicant would again point out that **Mellor-Crummy does not have the notion of threads as defined in a Java-like language** (e.g., see Footnote 1 on page 132). That is, unlike in a Java program, in Mellor-Crummy, only iterations of the same parallel loop are considered as candidates for datarace with one another.

Therefore, Applicant respectfully submits that Mellor-Crummey does not teach or suggest each and every element of the claimed invention. Therefore, the Examiner is respectfully requested to withdraw this rejection.

B. Flanagan

The Examiner alleges that Mellor-Crummey would have been combined with Flanagan to form the invention of claims 14-16 and 23. Applicant submits, however, that these references would not have been combined and even if combined, the alleged combination would not teach or suggested each and every element of the claimed invention.

Flanagan discloses a system for statically detecting potential race conditions in a multi-threaded computer program. In the system, a race condition detector determines whether accesses to object data fields are consistently protected by an appropriate lock. An object data field access that is not protected by an appropriate lock indicates a potential race condition (Flanagan at Abstract).

Applicant respectfully submits that these references would not have been combined as alleged by the Examiner. Indeed, these references are completely unrelated, and no person of ordinary skill in the art would have considered combining these disparate references, absent impermissible hindsight.

In fact, Applicant submits that these references do not include any motivation or suggestion to urge the combination as alleged by the Examiner. Indeed, these references clearly do not teach or suggest their combination.

Therefore, Applicant respectfully submits that one of ordinary skill in the art would not have been so motivated to combine the references as alleged by the Examiner. Therefore, the Examiner has failed to make a prima facie case of obviousness.

Moreover, Applicant submits that neither Mellor-Crummey, nor Flanagan, nor any alleged combination thereof, teaches or suggests "*processing the set of input information by comparing threads that may execute statements in a statement pair*", as recited, for example, in claim 1 and similarly recited in claim 23. As noted above, this feature allows the claimed method (e.g., and system) to more accurately detect a datarace condition (e.g., a condition where a datarace definitely will occur or may possibly occur) than in conventional methods (Application at page 6, lines 12-15).

Clearly, this feature is not taught or suggested by Flanagan. Indeed, Flanagan is clearly

unrelated to the claimed invention. In fact, the Examiner is not even attempting to rely on Flanagan as teaching this feature, but merely relies on Flanagan as allegedly teaching an object-oriented programming language.

Further, Applicant would point out that **neither Mellor-Crummy's work nor Flanagan's work mentions threads as an explicit programming entity computed and associated with each statement in the program** (e.g., see Flanagan at col.4, lines 15-27, and col:5, lines 16-28). In addition, Flanagan, does not apply may- or must-alias analysis for identifying synchronization objects or references.

Thus, Flanagan is clearly unrelated to the claimed invention, in which processes input information by comparing threads that may execute statements in a statement pair. Thus, Flanagan clearly does not make up for the deficiencies of Mellor-Crummey.

Therefore, Applicant submits that these references would not have been combined and even if combined, the combination would not teach or suggest each and every element of the claimed invention. Therefore, the Examiner is respectfully requested to withdraw this rejection.

III. FORMAL MATTERS AND CONCLUSION

In view of the foregoing, Applicant submits that claims 1-23, all the claims presently pending in the application, are patentably distinct over the prior art of record and are in condition for allowance. The Examiner is respectfully requested to pass the above application to issue at the earliest possible time.

Should the Examiner find the application to be other than in condition for allowance, the Examiner is requested to contact the undersigned at the local telephone number listed below to discuss any other changes deemed necessary in a telephonic or personal interview.

The Commissioner is hereby authorized to charge any deficiency in fees or to credit any overpayment in fees to Assignee's Deposit Account No. 50-0510.

Respectfully Submitted,

Date: 7/29/05

Phillip E. Miller, Esq.
Registration No. 46,060

McGinn & Gibb, PLLC
8321 Old Courthouse Road, Suite 200
Vienna, VA 22182-3817
(703) 761-4100
Customer No. 21254

CERTIFICATE OF FACSIMILE TRANSMISSION

I hereby certify that the foregoing Amendment was filed by facsimile with the United States Patent and Trademark Office, Examiner Chrystine Pham, Group Art Unit # 2192 at fax number (571) 273-8300 this 29th day of July, 2005.

Phillip E. Miller
Reg. No. 46,060